Когда "еще один пуллинг каждый N секунд" стучится вам в код. Время подумать про вебсокеты a.k.a полнодуплексное соединение.

Речь пойдет про socket.io , не совсем web socket а скорее микс при участии web socket. Но очень удобный в использовании сразу из коробки.

Кейс состоит в следующем:

Есть многопользовательское приложение в котором пользователи запускают асинхронные операции на сервере. Другими словами нажимают на кнопку в приложении и ждут когда сервер выполнит все операции а походу еще и рассекают про текущее состояние.

Вроде бы можно обойтись лонг пуллингом, но некоторые действия хочется заблокировать пользователям которые находятся за другими мониторами на той же странице. Да и вообще говоря сама библиотека при отсутствии возможности ws/wss соединения будет использовать пуллинг. Так что вроде бы только плюсы, из минусов только еще один пакет на клиенте и сервере.

Для клиента необходима библиотека socket-io.client, для фронта все написано на React и само соединение можно установить/разорвать через хуки. Для тех кто использует redux-toolkit все можно сделать через RTK.

У вас должен быть базовый объект createApi({ ... }) который описывает все эндпоинты и который потом, так же можно расширить.

```
import { createApi, fetchBaseQuery } from '@reduxjs/toolkit/query/react';

export const baseApi = createApi({
    reducerPath: 'baseApi',
    baseQuery: fetchBaseQuery({ baseUrl: '/' }),
    endpoints: () => ({}),
});
```

Добавляем в корневое состояния приложения новый редьюсер, если его конечно нет.

```
const appReducer = combineReducers({
  ...,
[baseApi.reducerPath]: baseApi.reducer,
  ...
});
```

В middleware добавляем => `baseApi.middleware`

И расширяем свой базовый baseApi функционалом для создания вебсокета.

```
import { baseApi } from '../api';
import { io } from 'socket.io-client';

export const wsApi = baseApi.injectEndpoints({
    endpoints: build => ({
        subscribeToEvents: build.query<any, void>({
            queryFn: () => ({ data: [] }),
            async onCacheEntryAdded(_arg, { dispatch, updateCachedData, cacheEntryRemoved }) {
                // Path is a prefix that will be used right after domain name
                const socket = io(`${your_url}/events`, {
                    path: '/socket.io',
                });

                socket.on('disconnect', reason => {
                    if (reason === 'io server disconnect') {
                        // the disconnection was initiated by the server, you need to reconnect manually
                        socket.connect();
                    }
                    // else the socket will automatically try to reconnect
                });

                socket.on('EVENT_TYPE', (event: ServerEvent) => {
                    // Here we should add the logic
                    updateCachedData(draft => {
                        draft.push(event);
                    });
                });

                await cacheEntryRemoved;
                socket.close();
            },
        }),
    }),
    overrideExisting: false,
});

export const { useSubscribeToEventsQuery } = wsApi;
```

injectEndpoints работает как раз для расширения эндпоинтов, только не забудьте добавить overrideExisting: false чтобы расширить а не переопределить существующий функционал.

useSubscribeToEventsQuery(); можно использовать как обычный хук, в том компоненте в котором желаете подписаться на события, например в App. Еще в api есть свойство keepUnusedDataFor для того чтобы задать время в секундах существования подключения/данных после последнего unsubscribe, по умолчанию 60 секунд.

На сервере используется Ts.Ed (Node js), но также существуют готовые библиотеки на других языках. Нужно проинсталлировать пакеты связанные с socket.io для сервера. И дела за малым, добавить конфиг:

```
socketIO: {
  path: '/socket.io',
  cors: {
    origin: '*' // put your servers
  }
}
```

И добавить сервис который выполняет подключение/отключение клиента а так же отправляет и принимает сообщения.

```
import { IO, Nsp, Socket, SocketService, SocketSession } from "@tsed/socketio";
import * as SocketIO from "socket.io";

@SocketService("/events") // namespace right after path 'socket.io'
export class MySocketService {

    @Nsp nsp: SocketIO.Namespace | undefined;

    // a map to keep clients by any id you like, a userId or whatever.
    public clients: Map<string, SocketIO.Socket> = new Map();


    constructor(@IO private io: SocketIO.Server) {
    }

    /**
     * Triggered when a new client connects to the Namespace.
     */
    $onConnection(@Socket socket: SocketIO.Socket, @SocketSession session: SocketSession) {
        this.clients.set(socket.id, socket);
    }

    // setup a method to send data to all clients
    // you can use this from any other service or controller.
    broadcast(someData: any): void {
        this.nsp.emit('EVENT_TYPE', { ... }: ServerEvent);
    }

    // method to send to a targeted client
    sendToSingleClient(idToSendTo: string, someData: any): void {
        const socket = this.clients.get(idToSendTo);
        if (!socket) return;
        socket.emit('EVENT_TYPE', { ... }: ServerEvent);
    }

}
```

Namespace в socket.io это путь "/event" после основного пути в настройках socketIO: { path: 'socket.io' }. Теперь можно в любом необходимом месте заинжектить сервис и отправить сообщение клиентам.

Вообщем то как-то так, после этого клиент может получать сообщения и выполнять необходимые side effects.